A Hybrid of Genetic Algorithm and Particle Swarm Optimization for Recurrent Network Design

Chia-Feng Juang, *Member, IEEE*

***Abstract*—An evolutionary recurrent network which automates the design of recurrent neural/fuzzy networks using a new evolutionary learning algorithm is proposed in this paper. This new evolutionary learning algorithm is based on a hybrid of genetic algorithm (GA) and particle swarm optimization (PSO), and is thus called HGAPSO. In HGAPSO, individuals in a new generation are created, not only by crossover and mutation operation as in GA, but also by PSO. The concept of elite strategy is adopted in HGAPSO, where the upper-half of the best-performing individuals in a population are regarded as elites. However, instead of being reproduced directly to the next generation, these elites are first enhanced. The group constituted by the elites is regarded as a swarm, and each elite corresponds to a particle within it. In this regard, the elites are enhanced by PSO, an operation which mimics the maturing phenomenon in nature. These enhanced elites constitute half of the population in the new generation, whereas the other half is generated by performing crossover and mutation operation on these enhanced elites. HGAPSO is applied to recurrent neural/fuzzy network design as follows. For recurrent neural network, a fully connected recurrent neural network is designed and applied to a temporal sequence production problem. For recurrent fuzzy network design, a Takagi–Sugeno–Kang-type recurrent fuzzy network is designed and applied to dynamic plant control. The performance of HGAPSO is compared to both GA and PSO in these recurrent networks design problems, demonstrating its superiority.**

***Index Terms*—Dynamic plant control, elite strategy, recurrent neural/fuzzy work, temporal sequence production.**

I. Introduction

THE ADVENT OF evolutionary computation has inspired new resources for optimization problem solving, such as the optimal design of neural networks and fuzzy systems. In contrast to traditional computation systems which may be good at accurate and exact computation, but have brittle operations, evolutionary computation provides a more robust and efficient approach for solving complex real-world problems [1]–[3]. Many evolutionary algorithms, such as genetic algorithm (GA) [4], genetic programming [5], evolutionary programming [6], and evolution strategies [7], have been proposed. Since they are heuristic and stochastic, they are less likely to get stuck in local minimum, and they are based on populations made up of individuals with a specified behavior similar to biological phenomenon. These common characteristics led to the development of evolutionary computation as an increasing important field.

Manuscript received February 14, 2003; revised May 20, 2003. This work was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC 91-2213-E-055-022. This paper was recommended by Associate Editor L. O. Hall.

The author is with the Department of Electrical Engineering, National Chung Hsing University, Taichung, 402 Taiwan, R.O.C.

Digital Object Identifier 10.1109/TSMCB.2003.818557

Among existing evolutionary algorithms, the most well-known branch is GA. GAs are stochastic search procedures based on the mechanics of natural selection, genetics, and evolution [4]. Since they simultaneously evaluate many points in the search space, they are more likely to find the global solution of a given problem. In addition, they use only a simple scalar performance measure that does not require or use derivative information, so they are general-purpose optimization methods for solving search problems. Two major primary areas in which GAs have been employed are optimization and machine learning. In machine learning, GAs have been successfully applied to the learning of neural networks [9], [10] and fuzzy systems [11], [12].

Recently, a new evolutionary computation technique, the particle swarm optimization (PSO), is proposed [13], [14]. Like GA, PSO is initialized with a population of random solutions. Its development was based on observations of the social behavior of animals such as bird flocking, fish schooling, and swarm theory. Each individual in PSO is assigned with a randomized velocity according to its own and its companions' flying experiences, and the individuals, called particles, are then flown through hyperspace. Compared with GA, PSO has some attractive characteristics. It has memory, so knowledge of good solutions is retained by all particles; whereas in GA, previous knowledge of the problem is destroyed once the population changes. It has constructive cooperation between particles, particles in the swarm share information between them. Successful applications of PSO to several optimization problems, like function minimization [15] and feedforward neural network design [16]–[18], have demonstrated its potential.

Temporal information processing problems are encountered in many areas, such as control, communication, and pattern recognition. To solve this type of problems, one effective approach is the use of recurrent networks. Recurrent networks have self-loops and backward connections in their topologies, and these feedback loops are used to memorize past information. Therefore, they can be used to deal with temporal/spatialtemporal problems. Many types of recurrent networks have been proposed, among which two widely used categories are recurrent neural networks [19] and recurrent fuzzy networks [20]–[27], each of which has its suitable application domain. For temporal sequence production problems, like the generation of limit cycles and chaos, recurrent neural networks are effective. In contrast, for problems that include concurrent spatial and temporal mapping, the performance of recurrent fuzzy network has shown to outperform that of recurrent neural networks [26], [27]. In using recurrent networks, the network parameters must be decided. Many supervised learning algorithms have been proposed for recurrent network training, like back propagation through time (BPTT) [28], real-time recurrent learning (RTRL) [29], and time-dependent recurrent back propagation (TDBR) [30]. These algorithms are based on gradient descent, are complex in derivation, easily trapped at local minima, and are only suitable to problems where desired output values are available. Another parameter design approach is by GA [40]–[31], which is suitable to problems where desired outputs are unavailable or costly to obtain. However, the learning performance of GA may be unsatisfactory for complex problems. In addition, for the learning of recurrent network weights, many possible solutions exist. Two individuals with high fitness values are likely to have dissimilar set of weights, and the recombination may result in offspring with poor performance. This type of problem is referred to as the structural/functional mapping problem [33]. For these problems, we propose recurrent networks designed by a new algorithm, the HGAPSO.

Hybridization of evolutionary algorithms with local search has been investigated in many studies [34]–[36]. Such a hybrid is often referred to as a memetic algorithm [37]–[39]. In contrast to memetic algorithm, we will combine two global optimization algorithms, i.e., GA and PSO, in this paper. Since PSO and GA both work with a population of solutions, combining the searching abilities of both methods seems to be a good approach. Originally, PSO works based on social adaptation of knowledge, and all individuals are considered to be of the same generation. On the contrary, GA works based on evolution from generation to generation, so the changes of individuals in a single generation are not considered. Based on the compensatory property of GA and PSO, we propose a new algorithm that combines the evolution ideas of both. In the reproduction and crossover operation of GAs, individuals are reproduced or selected as parents directly to the next generation without any enhancement. However, in nature, individuals will grow up and become more suitable to the environment before producing offspring. To incorporate this phenomenon into GA, PSO that is inspired by social interaction of knowledge is adopted to enhance the top-ranking individuals on each generation. In a social science context, PSO enhances individuals by both sharing information between each other and their individually learned knowledge. Then, these enhanced individuals are reproduced and selected as parents for crossover operation. Offsprings produced by the enhanced individuals are expected to perform better than some of those in original population, and the poor-performed individuals will be weeded out from generation to generation. To demonstrate the searching ability of HGAPSO, designs of recurrent networks, including fully-connected recurrent neural network (FCRNN) and Takagi–Sugeno–Kang (TSK)-type recurrent fuzzy network (TRFN), are simulated.

This paper is organized as follows. In Section II, the architectures of recurrent networks, including FCRNN and TRFN, are introduced. Overviews of GAs and PSO are described in Section III. Section IV describes the learning algorithm of HGAPSO for recurrent network design. In Section V, designs of FCRNN and TRFN by HGAPSO are simulated and applied to temporal sequence production and dynamic plant control, respectively. Finally, conclusions are presented in Section VI.

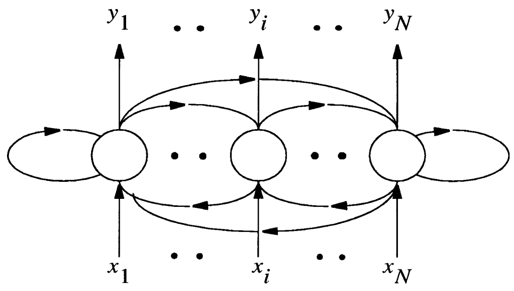

Fig. 1. Architecture of the fully connected recurrent neural network (FCRNN).

II. Recurrent Networks

In this section, the recurrent neural network and the recurrent fuzzy network to be designed by HGAPSO are introduced. The recurrent neural network to be designed is the fully connected recurrent neural network (FCRNN), as introduced in Ssection II-A. In Section II-B, a recurrent fuzzy network, the TSK-type recurrent fuzzy network (TRFN) is introduced.

A. Fully Connected Recurrent Neural Network (FCRNN)

The architecture of the FCRNN is shown in Fig. 1. In FCRNN, each node is connected to itself and to all other nodes. Individual nodes may be input nodes, output nodes, neither or both. The desired outputs $d_i(t)$ are usually defined only on certain nodes at certain times. Let x_i and y_i denote the node input and output, respectively. The nodes in FCRNN evolve according to

$$\tau_i \dot{y}_i = -y_i + a\left(\sum_j w_{ij} y_j - 0.5\right) + x_i \tag{1}$$

where τ_i is the relaxation time scale, and the activation function a used here is a unipolar sigmoid function, i.e., $a(h) = 1/(1+\exp(-h))$. In computer simulation, the derivative in (1) is approximated by first-order difference, and the equation becomes

$$y_i(t+\Delta t) = \left(1 - \frac{\Delta t}{\tau_i}\right) y_i(t) + \frac{\Delta t}{\tau_i} a\Big(h_i(t)\Big) + \frac{\Delta t}{\tau_i} x_i(t) \tag{2}$$

where $h_i(t) = \sum_j w_{ij} y_j(t) - 0.5$ is the net input to node i. In [40], [41], designs of FCRNN by GAs for temporal sequence production are performed. However, in this current paper, we perform the same task by HGAPSO and demonstrate its superiority.

B. TSK-Type Recurrent Fuzzy Network (TRFN)

The TRFN to be designed is constructed from a series of recurrent fuzzy if-then rules with TSK-type consequent parts [27]. In [27], the superiority of TRFN to recurrent neural network in spatial-temporal problems, including dynamic plant identification and control, was demonstrated. The structure of TRFN is shown in Fig. 2. Nodes in layer 1 are input nodes. The nodes in layer 2 act as membership functions to express the input fuzzy linguistic variables. Two types of membership functions are used in this layer. For the external variable x, a local membership function, the Gaussian membership function is adopted.

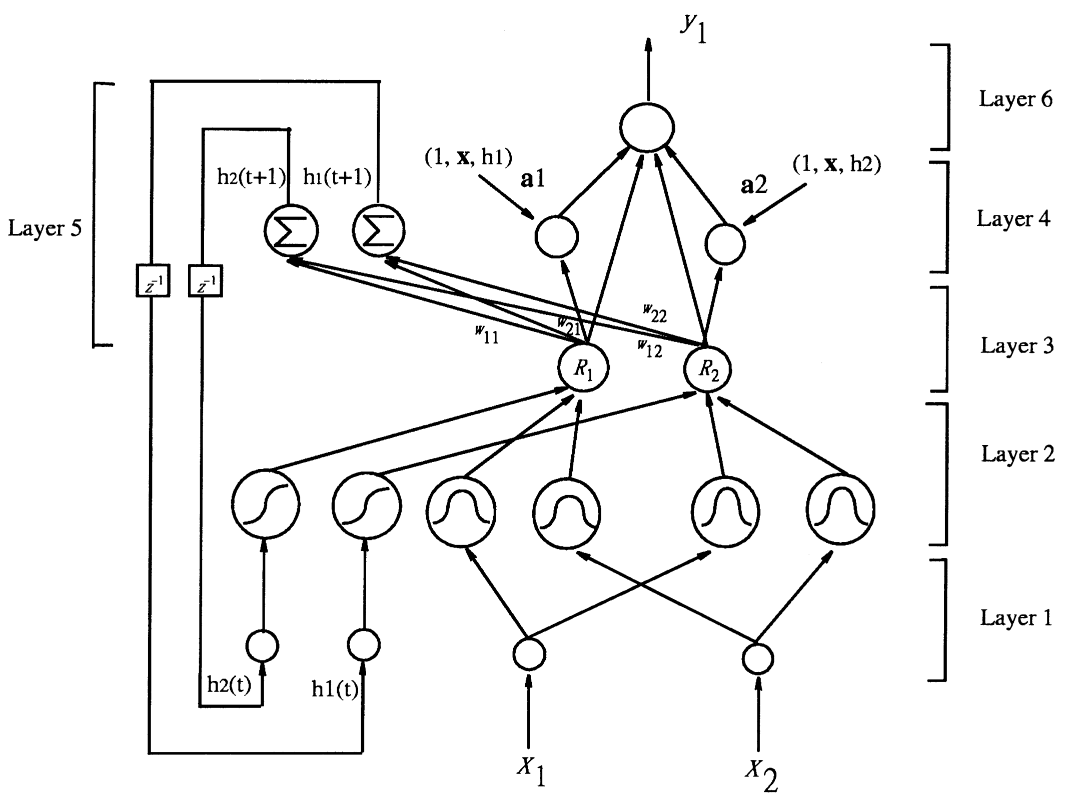

Fig. 2. Architecture of the TSK-type recurrent fuzzy network (TRFN).

For the internal variable h, a global membership function, the sigmoid function is adopted. Each internal variable has a single corresponding fuzzy set. Each node in layer 3 is a rule node, and nodes in layer 4 are called consequent nodes. Each rule node has a corresponding consequent node which performs a weighted linear combination of the input variables x and h plus a constant. Nodes in layer 5 are called context nodes and perform the defuzzification operation. The number of internal variables h in this layer is equal to the rule nodes. In layer 6, the node is called a defuzzification node. In TRFN, the recurrent property comes from feeding the internal variables, derived from fuzzy firing strengths, back to both the network input and output layers. In this configuration, each internal variable is responsible for memorizing the temporal history of its corresponding fuzzy rule. The internal variable is also combined with external input variables in each rules consequence.

Each recurrent fuzzy if-then rule in TRFN, consisting of N_r rules and n external inputs, $x_1 \sim x_n$, is in the following form:

$$\begin{aligned}&\text{Rule } i\text{: IF } x_1(t) \text{ is } A_{i1} \text{ and } x_2(t) \text{ is } A_{i2} \ldots \text{ and}\\&\quad x_n(t) \text{ is } A_{\text{in}} \text{ and } h_i(t) \text{ is } G \text{ THEN}\\&\quad y(t+1) \text{ is } a_{i0} + a_{i1}x_1(t)\\&\quad + \cdots + a_{\text{in}}x_n(t) + a_{\text{in}+1}h_i(t)\\&\quad \text{and } h_1(t+1) \text{ is } w_{i1} \text{ and } h_2(t+1)\\&\quad \text{is } w_{i2} \ldots \text{ and } h_{N_r}(t+1) \text{ is } w_{iN_r}\end{aligned} \tag{3}$$

where A and G are fuzzy sets, w and a are the consequent parameters for inference output h and y, respectively. The consequent part for the external output y is of TSK-type and is a linear combination of the external input variables x and internal variables h, plus a constant. The recurrent reasoning implies that the inference output $y(t+1)$ is affected by the internal variable $h_i(t)$, and the current internal output $h_i(t+1)$ is a function of previous output value $h_i(t)$; i.e., the internal variable h_i itself forms a recurrent reasoning. In TRFN structure, if there are N_r rules, then the numbers of internal variables h and feedback weights connecting to each internal variables are also equal to N_r.

In TRFN, a Gaussian membership function is used for the fuzzy set A. For the fuzzy set G, a global membership function $G(x) = 1/(1+e^{-x})$ is used. Given an input set $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, the inference and internal output of TRFN are calculated, respectively, by

$$y(t+1) = \frac{\sum_{i=1}^{N_r} \mu_i\Big(\mathbf{x}(t), h_i(t)\Big) f_i(t)}{\sum_{i=1}^{N_r} \mu_i(\mathbf{x}(t), h_i(t))} \tag{4}$$

$$h_i(t+1) = \sum_{k=1}^{N_r} \mu_k\Big(\mathbf{x}(t), h_i(t)\Big) w_{ik} \tag{5}$$

where

$$f_i(t) = a_{i0} + \sum_{j=1}^{n} a_{ij}x_j(t) + a_{in+1}h_i$$

$$\mu_i\Big(\mathbf{x}(t), h_i(t)\Big) = \frac{1}{1+e^{-h_i(t)}} \cdot \exp\left\{-\sum_{j=1}^{n}\left(\frac{x_j(t)-m_{ij}}{\sigma_{ij}}\right)^2\right\}.$$

In [27], TRFN design by GA is performed. Here, we will further apply HGAPSO to TRFN design and demonstrate the superiority of HGAPSO to GA.

III. GAs and PSO

The proposed HGAPSO combines GA with PSO to form a hybrid GA [4]. This hybrid of a GA with existing algorithms can always produce a better algorithm than either the GA or the existing algorithms alone [43]–[46]. In this section, basic concepts of GAs and PSO are introduced, followed by a detailed introduction of HGAPSO in the next section.

A. Basic Concepts of GAs

In GA, a candidate solution for a specific problem is called an individual or a chromosome and consists of a linear list of genes. Each individual represents a point in the search space, and hence a possible solution to the problem. A population consists of a finite number of individuals. Each individual is decided by an evaluating mechanism to obtain its fitness value. Based on this fitness value and undergoing genetic operators, a new population is generated iteratively with each successive population referred to as a generation. The GAs use three basic operators (reproduction, crossover, and mutation) to manipulate the genetic composition of a population. Reproduction is a process by which the most highly rated individuals in the current generation are reproduced in the new generation. The crossover operator produces two offsprings (new candidate solutions) by recombining the information from two parents. There are two processing steps in this operation. In the first step, a given number of crossing sites are selected uniformly, along with the parent individual at random. In the second step, two new individuals are formed by exchanging alternate pairs of selection between the selected sites. Mutation is a random alteration of some gene values in an individual. The allele of each gene is a candidate for mutation, and its function is determined by the mutation probability. Many efforts on the enhancement of traditional GAs have been proposed [47]. Among them, one category focuses on modifying the structure of the population or the role an individual plays in it [48]–[51], such as distributed GA [49], cellular GA [50], and symbiotic GA [51]. Another category aims to modify the basic operations, such as crossover or mutation, of traditional GAs [52]–[54].

B. Basic Concepts of PSO

The PSO conducts searches using a population of particles which correspond to individuals in GA. A population of particles is randomly generated initially. Each particle represents a potential solution and has a position represented by a position vector $\vec{x}_i$. A swarm of particles moves through the problem space, with the moving velocity of each particle represented by a velocity vector $\vec{v}_i$. At each time step, a function f_i representing a quality measure is calculated by using $\vec{x}_i$ as input. Each particle keeps track of its own best position, which is associated with the best fitness it has achieved so far in a vector $\vec{p}_i$. Furthermore, the best position among all the particles obtained so far in the population is kept track of as $\vec{p}_g$. In addition to this global version, another local version of PSO keeps track of the best position among all the topological neighbors of a particle.

At each time step t, by using the individual best position, $\vec{p}_i(t)$, and global best position, $\vec{p}_g(t)$, a new velocity for particle i is updated by

$$\vec{v}_i(t+1) = \vec{v}_i(t) + c_1\phi_1(\vec{p}_i(t) - \vec{x}_i(t)) + c_2\phi_2(\vec{p}_g(t) - \vec{x}_i(t)) \quad (6)$$

where c_1 and c_2 are positive constants and ϕ_1 and ϕ_2 are uniformly distributed random numbers in [0, 1]. The term $\vec{v}_i$ is limited to the range $\pm\vec{v}_{\max}$. If the velocity violates this limit, it is set at its proper limit. Changing velocity this way enables the particle i to search around its individual best position, $\vec{p}_i$, and global best position, $\vec{p}_g$. Based on the updated velocities, each particle changes its position according to the following:

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1). \quad (7)$$

Based on (6) and (7), the population of particles tend to cluster together with each particle moving in a random direction. The computation of PSO is easy and adds only a slight computation load when it is incorporated into GA. Furthermore, the flexibility of PSO to control the balance between local and global exploration of the problem space helps to overcome premature convergence of elite strategy in GA, and also enhances searching ability. In the next section, the detailed algorithm for the hybrid of PSO with GA is introduced.

IV. Hybrid of GA and PSO (HGAPSO)

The detailed design algorithm of recurrent networks by HGAPSO consists of three major operators: enhancement, crossover, and mutation. Before describing details of these operators, the issues of coding and initialization are presented. Coding concerns the way the weights of recurrent networks are represented by individuals, whereas initialization is the proper assignment of learning constants before entering the evolution process. The overall learning process is described step by step below.

- Coding. A floating point coding scheme is adopted here. For FCRNN coding, suppose there are N nodes, then the total number of weights to be coded is N^2; that is, weights $w_{11} \sim w_{NN}$ should be coded into an individual. The coding of a FCRNN into an individual is as follows:

  $$|w_{11}|w_{21}|\cdots|w_{N1}|w_{12}|\cdots|w_{N-1N}|w_{NN}|.$$

  For TRFN, the free parameters to be coded include m, σ, w, and a. Instead of using a grid type partition in the input space as in most existing genetic fuzzy systems [56], [57], a flexible external input space partition is adopted here, so the number of membership functions A on each external input variables x is also equal to the total rule number N_r. And for coding uniformity in different application problems, the value of each external input data x is scaled to be in the range between -1 and 1. After scaling, the initial values of m and σ are randomly assigned within their corresponding searching ranges $[-1, 1]$ and $[0.05, 0.75]$, respectively. Within the searching range, there is no restriction on

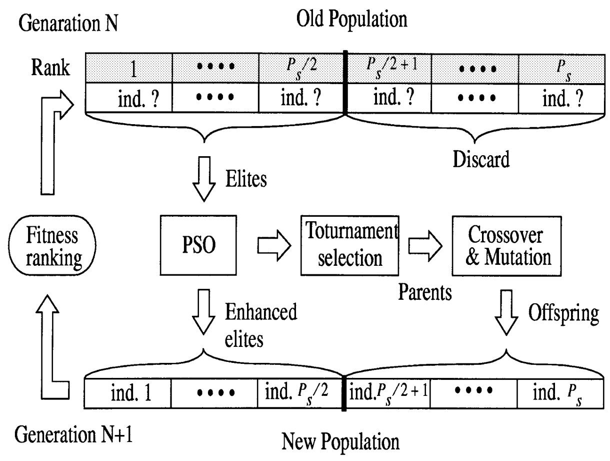

Fig. 3. Flow of HGAPSO.

the position and width of the membership function. Geometrically, for each individual, the N_r randomly generated rules correspond to N_r clusters in the spatial input space. These clusters may be located in arbitrary positions in the input space region and a flexible partition is achieved. Suppose there is single output in TRFN, then each individual has the following form:

$$|m_{11}|\sigma_{11}|\cdots|m_{1n}|\sigma_{1n}|a_{10}|\cdots|a_{1n+1}|m_{21}| \cdots|a_{N_rn+1}|w_{11}|\cdots|w_{N_r1}|\cdots|w_{N_rN_r}|$$

where there is a total of $N_r(N_r+3n+2)$ genes. The order of parameters coded into the individual is as follows. First, for rule 1, the mean m and standard variation σ of input variables 1 to n are coded, followed by the consequent part parameters $a_{10} \sim a_{1n+1}$. Following rule 1, rules 2 to N_r are sequentially coded in the same way. Up to now, the parameters representing the spatial relation are coded. Next, we code the temporal relations into the individuals. First, weights $w_{11} \sim w_{N_r1}$ connected to the same hidden variable h_1 are coded, followed by the weights connecting to h_2 to h_{N_r}. The total number of feedback weights coded is N_r^2.

- Initialization. In HGAPSO, GA and PSO both work with the same population. Initially, P_s individuals forming the population should be randomly generated. These individuals may be regarded as chromosomes in terms of GA, or as particles in terms of PSO. In FCRNN, the number of neural nodes, N, and in TRFN, the total number of rules, N_r, should be assigned in advance. In addition, the learning parameters, such as c_1 and c_2 in PSO, and the mutation probability p_m should be assigned in advance. After initialization, new individuals on the next generation are created by enhancement, crossover, and mutation operations. For clatity, the flow of these operations is illustrated in Fig. 3.
- *Enhancement.* In each generation, after the fitness values of all the individuals in the same population are calculated, the top-half best-performing ones are marked. These individuals are regarded as elites. Instead of reproducing the elites directly to the next generation as elite GAs do, we first enhance the elites. The enhancement operation tries to mimic the maturing phenomenon in nature, where individuals will become more suitable to the environment after acquiring knowledge from the society. Furthermore, by using these enhanced elites as parents, the generated offsprings will achieve better performance than those bred by original elites. Then there is the problem of how to enhance individuals of the same generation, which is handled by PSO. Here, the group constituted by the elites may be regarded as a swarm, and each elite corresponds to a particle in it. In PSO, individuals of the same generation enhance themselves based on their own private cognition and social interactions with each other. In HGAPSO, we adopt and regard this technique as the maturing phenomenon. Based on PSO, (6) and (7) may be applied to the elites. Here, instead of using (6), the following modified equation [15] is adopted

$$\vec{v}_i(t+1) = \chi\Big(\vec{v}_i(t) + c_1\phi_1\big(\vec{p}_i(t) - \vec{x}_i(t)\big) + c_2\phi_2\big(\vec{p}_g(t) - \vec{x}_i(t)\big)\Big) \quad (8)$$

where χ controls the magnitude of $\vec{v}$. With parameter χ, no explicit limit $\vec{v}_{\max}$ is required. Similar idea is also proposed in [55], where a new parameter, called inertia weight, is introduced into (6). Here, $\vec{p}_g(t)$ is the best-performing individual evolved so far, either the enhanced elite or the produced offspring. If the elite i is the offspring produced by the parents of a previous generation, then $\vec{v}_i(t)$ is set to zero, and $\vec{p}_i(t)$ is set to $\vec{x}_i(t)$, i.e., the newly generated individual itself. Otherwise, $\vec{p}_i(t)$ records the best solution of individual i evolved so far. Equation (8) combines a cognition-only model and a social-only model [42] for individual enhancement. The term $\phi_1(\vec{p}_i(t) - \vec{x}_i(t))$ represents a cognition-only model, where individuals are treated as isolated beings and cognition occurs privately. On the other hand, the term $\phi_2(\vec{p}_g(t) - \vec{x}_i(t))$ represents a social-only model, where the private knowledge of individuals is ignored. Rather, individuals change according to the successful benefits of neighborhood members. After performing (7), it should be noted that the value of the individual may exceed its reasonable range. For this case, we should reassign the value to its nearest reasonable value. For instance, in TRFN design, when the width σ tuned by (7) is smaller than the minimum reasonable width $\sigma_{\min} = 0.05$, we should assign its width to $\sigma_{\min}$. By performing PSO on the elites, we may avoid the premature convergence in elite GAs and increase the search ability. Half of the population in the next generation are occupied by the enhanced individuals, the others by crossover operation.
- *Crossover.* To produce well-performing individuals, in the crossover operation parents are selected from the enhanced elites only. To select parents for the crossover operation, the tournament selection scheme is used, in which two enhanced elites are selected at random, and their fitness values are compared to select the elite with better fitness value as one parent. Then the other parent is selected in the same way. Two offsprings are created by performing crossover on the selected parents. Some of the commonly used crossover techniques are k-point crossover, where k crossover sites are selected randomly

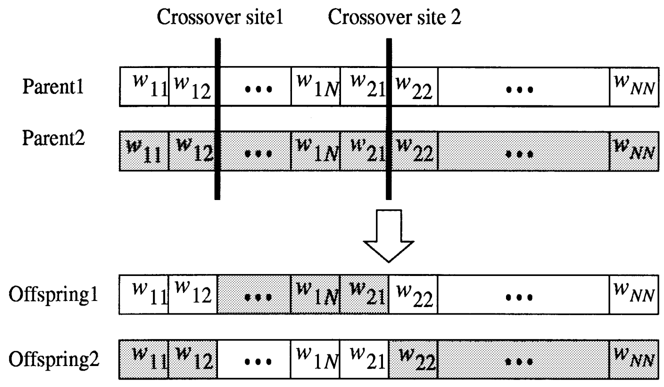

Fig. 4. Crossover operation on the individuals encoding the FCRNN.

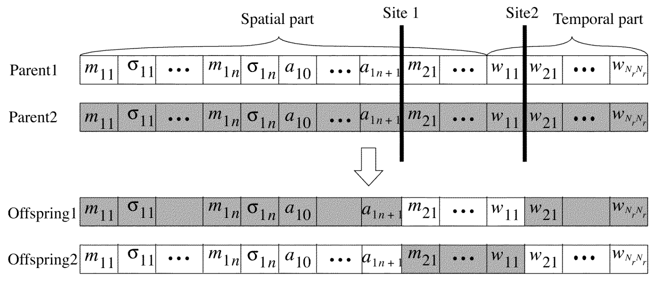
Spatial part
Site 1
Site2
Temporal part
Parent1
Parent2
Offspring1
Offspring2
m_{11} σ_{11} ••• m_{1n} σ_{1n} a_{10} ••• a_{1n+1} m_{21} ••• w_{11} w_{21} ••• $w_{N_rN_r}$

Fig. 5. Crossover operation on the individuals encoding the TRFN.

within the range of an individual and swapping occurs; and flat crossover, which produces offspring using a linear combination of two parents. In this study, two point crossover operation is used. In FCRNN, the two crossover sites are selected at random. Then, the parents are crossed and separated at the two sites as shown in Fig. 4. For TRFN, one crossover site is randomly located at the gene position coded for spatial rule relations, the other at the position for temporal rule consequences as shown in Fig. 5. These produced offsprings occupy half of the population in the next generation. The adopted crossover may be regarded as a kind of elite crossover and is used to increase the searching ability. As in nature, offspring bred by superior parents will achieve better fitness than those by average parents. From the perspective of PSO, where individuals work on the same generation, crossover operation in HGAPSO introduces the concepts of generational evolution and survival of the fitest in society.

- *Mutation.* In HGAPSO, mutation occurs in conjunction with the crossover operation. Hence, mutation is an operator whereby the allele of a gene is altered randomly so that new genetic materials can be introduced into the population. However, mutation should be used sparingly because it is a random search operator; otherwise, with high mutation rates, the algorithm will become little more than a random search. Here, uniform mutation is adopted, that is, the mutated gene is drawn randomly, uniformly from the corresponding search interval. In the following simulations, a constant mutation probability $p_m = 0.1$ is used. Other adaptive mutation strategies, like those seen in [53], can also be adopted to further improve the performance.

V. Simulations

In this section, HGAPSO is applied to FCRNN and TRFN design. For FCRNN design, the objective is to produce a temporal sequence; whereas for TRFN design, the objective is to control a dynamic plant. To demonstrate the superiority of HGAPSO, in each simulation, the performance of HGAPSO is compared with GA and PSO.

A. Design of FCRNN

In this section, HGAPSO is used to train a FCRNN for temporal sequence generation.

Example 1: For the FCRNN to be designed, there is no external input; that is, $x_i(t) = 0$ in (1). The initial values of $y_i(0)$ are set to 0.5. The relaxation time scale τ_i is set to 1, and the discrete-time step $\Delta t = 0.1$ is used. The FCRNN with no input node, 23 hidden nodes, and two output nodes, i.e., $N = 25$, all fully connected, is trained to produce the following two trajectories in the time interval $(t_0, t_1] = (4, 20]$

$$y_{r1}(t) = 0.35\sin(0.5t)\sin(1.5t)$$
$$y_{r2}(t) = 0.35\cos(0.5t)\sin(1.5t).$$

Hence, the network output state is required to be on the desired trajectory at $t_0 = 4$ and follow the desired trajectory afterwards. The outputs of the two output nodes are denoted as $y_{24}(t)$ and $y_{25}(t)$, respectively. During training, the root mean square error (RMSE) in time interval (4, 20] is calculated by

$$\mathrm{RMSE} = \left(\left(\sum_{k=1}^{160}\left(\left(y_{r1}(4+k\Delta t) - y_{24}(4+k\Delta t)\right)^2 + \left(y_{r2}(4+k\Delta t) - y_{25}(4+k\Delta t)\right)^2/160\right)\right)\right)^{1/2} \quad (9)$$

and the fitness value is defined to be $1/\mathrm{RMSE}$.

In applying HGAPSO, initially, 200 individuals are randomly generated in a population, i.e., $P_s = 200$. There are 25×25 genes in each individual, and the initial weights w_{ij} are uniform random values between 10 and -10. The parameters c_1, c_2, and χ are set to 1, 1, and 0.8, respectively. The evolution is processed for 1200 generations and is repeated for 50 runs. The averaged best-so-far RMSE value over 50 runs for each generation is shown in Fig. 6. The best and averaged RMSEs for the 50 runs after 1200 generations of training are listed in Table I. To demonstrate the learned result, the two trajectories generated by FCRNN are shown in Fig. 7. From Fig. 7, we see that FCRNN can learn and generate the two periodic trajectories even after $t_1 = 20$, where training is not performed.

To show the effectiveness and efficiency of HGAPSO, FCRNN designed by GA and PSO are applied to the same problem. In addition, one gradient descent based learning algorithm, the TDRB, is simulated. In GA, the population size and initial individuals are the same as those used in HGAPSO. The parents for crossover are selected from the whole population instead of from only the elites, and the tournament selection is used. The elite strategy is used, where the best individual of each generation is copied into the succeeding generation. Two crossover probabilities, $p_c = 0.5$ and $p_c = 0.8$, are simulated,

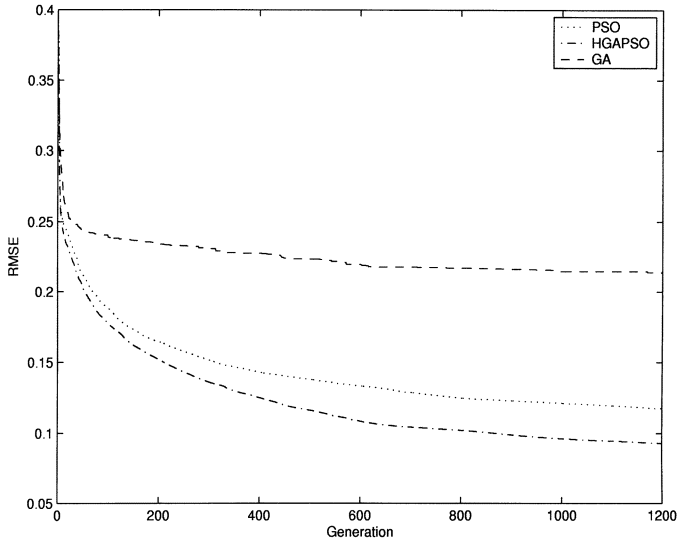

Fig. 6. Averaged best-so-far RMSE in each generation(iteration) for HGAPSO (-.), GA (- -), and PSO (...) in Example 1.

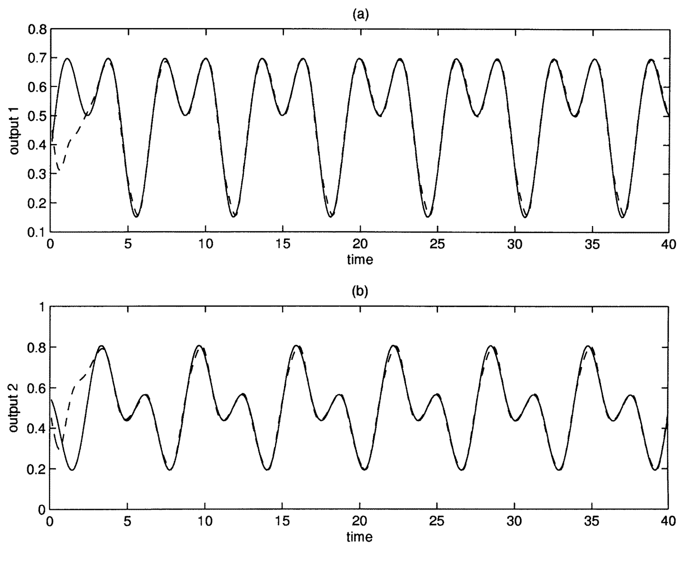

Fig. 7. Desired (solid line) and actual temporal sequence (–) generated by FCRNN designed by HGAPSO in Example 1. (a) Temporal sequence 1. (b) Temporal sequence 2.

TABLE I
PERFORMANCE COMPARISONS FOR DIFFERENT METHODS OF FCRNN DESIGN FOR TEMPORAL SEQUENCE PRODUCTION IN EXAMPLE 1

Method	Example 1				
	TDRR	GA (Pc=0.5)	GA (Pc=0.8)	PSO	HGAPSO
RMSE(Ave)	——	0.2138	0.2209	0.1175	0.0928
RMSE(Best)	0.0382	0.1684	0.1883	0.0172	0.0170

respectively. Like HGAPSO, the mutation probability p_m is 0.1. The design results for both crossover probabilities after 50 runs are listed in Table I. The averaged best-so-far RMSE for each generation with $p_c = 0.5$ is shown in Fig. 6. From Table I, we see that both the averaged and best RMSEs of HGAPSO are smaller than those of GA.

In PSO, as in HGAPSO, (8) is adopted. The population size and initial individuals are the same as those used in HGAPSO. For fair comparison, the parameters c_1, c_2, and χ are also the same as those used in HGAPSO. The evolution is processed for 1200 iterations, and the design results after 50 runs are listed in Table I. The average of the best RMSE achieved by solution $\vec{p}_g$ on each iteration, corresponding to generation in HGAPSO, is also shown in Fig. 6. From the comparison results, we see that the averaged and best RMSEs of PSO are smaller than those of GA, but are still larger than those of HGAPSO.

Finally, TDRB is applied to train the FCRNN. The learning constant η is set to 0.3. The iteration is 10000, and the best training result is listed in Table I. From Table I, we see that HGAPSO achieves the best results among all compared methods.

B. Design of TRFN

Dynamic system control by TRFN designed by HGAPSO is simulated in this subsection. For the dynamic system control problem, since the precise controller input-output training data is either costly to obtain or unavailable, the HGAPSO is adopted for controller design. Two examples, one for muti-input single output (MISO), the other for multi-input multi-output (MIMO) plant control, are simulated.

Example 2 (MISO Control): The controlled plant is the same as that used in [58], [27] and is given by

$$y_p(k+1) = \frac{y_p(k)y_p(k-1)(y_p(k)+2.5)}{1+y_p^2(k)+y_p^2(k-1)} + u(k). \quad (10)$$

In designing TRFN controller, the desired output y_r is specified by the following 250 pieces of data,

$$\begin{aligned} y_r(k+1) &= 0.6y_r(k) + 0.2y_r(k-1) + r(k), \\ r(k) &= 0.5\sin(2\pi k/45) + 0.2\sin(2\pi k/15) \\ &\quad + 0.2\sin(2\pi k/90). \end{aligned}$$

The inputs to TRFN controller are $y_p(k)$ and $y_r(k)$ and the output is $u(k)$. There are four rules in TRFN, i.e., $N_r = 4$, resulting in a total of 48 free parameters. In applying HGAPSO, initially, 50 individuals are randomly generated in a population, i.e., $P_s = 50$, and each individual contains 48 genes. The probability of mutation, p_m, is 0.1; and the parameters c_1, c_2, and χ are set to 1, 1, and 0.8, respectively. The RMSE of these 250 tracking data is calculated by

$$\mathrm{RMSE} = \left(\sum_{k=1}^{250} \left(y_r(k+1) - y_p(k+1)^2 \right)/250 \right)^{1/2}. \quad (11)$$

The fitness value is defined as $1/\mathrm{RMSE}$. The evolution is processed for 100 generations and is repeated for 100 runs. The averaged best-so-far RMSE value over 100 runs for each generation is shown in Fig. 8. The best and averaged RMSE error for the 100 runs after 100 generations of training are listed in Table II. To demonstrate the control result, one control result is shown in Fig. 9.

To show the effectiveness and efficiency of HGAPSO, TRFN designed by GA and PSO are applied to the same control problem. The GA used is the same as that used in example 1. The population size is set to 50. Two crossover probabilities,

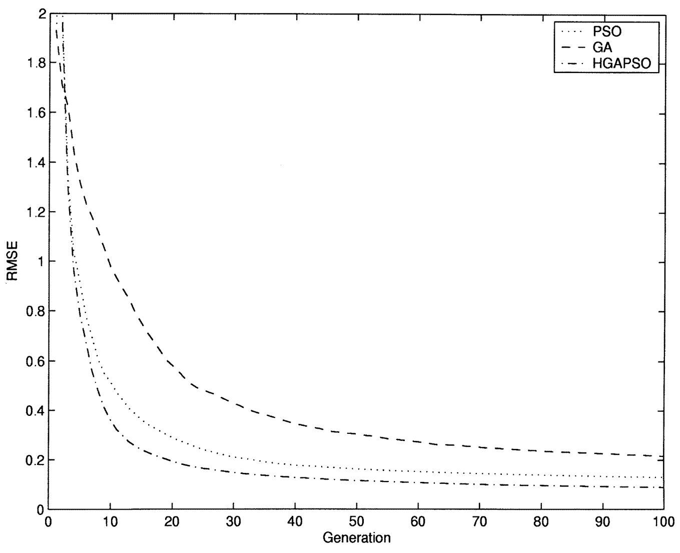

Fig. 8. Averaged best-so-far RMSE in each generation(iteration) for HGAPSO (-.), GA (- -), and PSO (...) in Example 2.

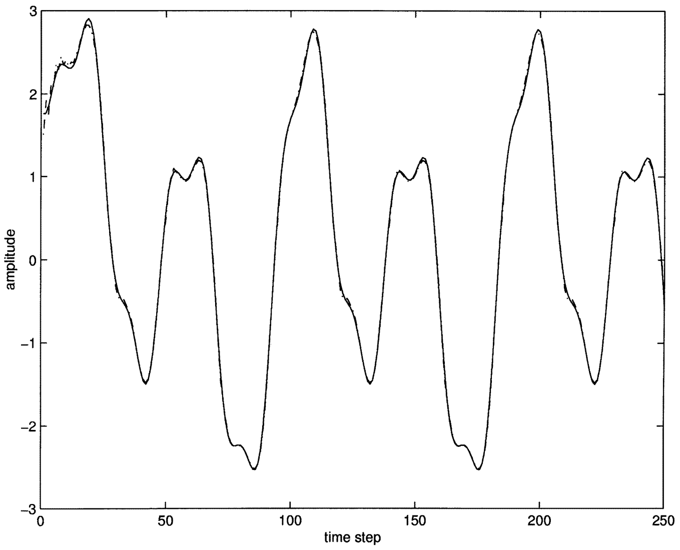

Fig. 9. Control performance of TRFN designed by HGAPSO in Example 2; the reference output is denoted by a solid curve and the actual output by a dotted curve.

TABLE II
PERFORMANCE COMPARISONS FOR DIFFERENT METHODS OF MISO TRFN CONTROLLER DESIGN FOR DYNAMIC PLANT CONTROL PROBLEM IN EXAMPLE 2

Method	Example 2			
	GA (Pc = 0.5)	GA (Pc = 0.8)	PSO	HGAPSO
RMSE(Ave)	0.2150	0.2704	0.1296	0.0890
RMSE(Best)	0.1040	0.1614	0.0503	0.0415

$p_c = 0.5$ and $p_c = 0.8$, are simulated, respectively. Like HGAPSO, the mutation probability p_m is 0.1. The design results after 100 runs are listed in Table II. The averaged best-so-far RMSE for each generation with $p_c = 0.5$ is shown in Fig. 10. From Table II, we see that both the averaged and best RMSEs of HGAPSO are obviously smaller than those of GA.

In PSO, the population size P_s, parameters c_1, c_2, and χ are the same as those used in HGAPSO. The design results after 100 runs are listed in Table II. The average of the best RMSE achieved by solution $\vec{p}_g$ on each iteration is also shown in Fig. 10. From the comparison results, we see that, as in example 1, the averaged and best RMSEs of PSO are smaller than those of GA, but are still larger than those of HGAPSO.

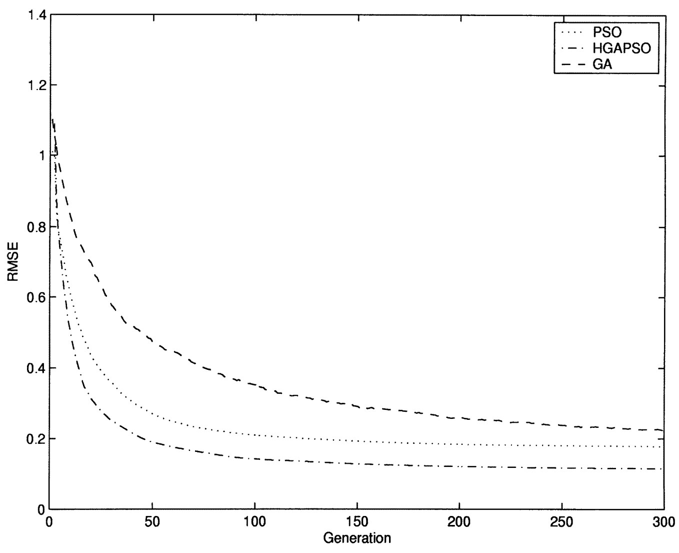

Fig. 10. Averaged best-so-far RMSE on each generation(iteration) for HGAPSO (-.), GA (- -), and PSO (...) in Example 3.

Example 3 (MIMO Control): The MIMO plant to be controlled is given by [58]

$$y_{p1}(k+1) = 0.5\left[\frac{y_{p1}(k)}{1+y_{p2}^2(k)} + u_1(k-1)\right]$$
$$y_{p2}(k+1) = 0.5\left[\frac{y_{p1}(k)y_{p2}(k)}{1+y_{p2}^2(k)} + u_2(k-1)\right].$$

The controlled outputs should follow desired outputs y_{r1} and y_{r2} as specified by the following 250 pieces of data:

$$y_{r1}(k) = \sin(k\pi/45)$$
$$y_{r2}(k) = \cos(k\pi/45).$$

The inputs to TRFN controller are $y_{p1}(k), y_{p2}, y_{r1}$, and $y_{r2}(k)$ and the outputs are $u_1(k)$ and $u_2(k)$. There are six rules in TRFN, i.e., $N_r = 6$, resulting in a total of 156 free parameters. Each individual contains 156 genes, and the parameters, p_m, c_1, c_2, and χ are the same as those used in example 2. The fitness value is defined as the inverse of the RMSE for both two outputs over these 250 tracking data. The evolution is processed for 300 generations and is repeated for 100 runs, and the averaged best-so-far RMSE value over 100 runs for each generation is shown in Fig. 10. The best and averaged RMSE error for the 100 runs after 300 generations of training are listed in Table III. To demonstrate the control result, one control performance is shown in Fig. 11.

To show the effectiveness and efficiency of HGAPSO, TRFN designed by GA and PSO are applied to the same control problem. The GA and PSO used are the same as those used in example 1, and the design results for both GA and PSO are listed in Table III. The averaged best-so-far RMSE for GA with $p_c = 0.5$ and PSO are shown in Fig. 11. From Table III, we

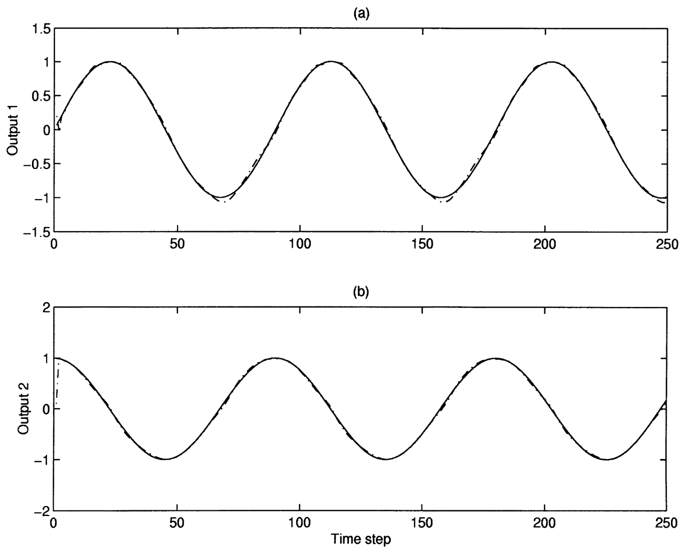

Fig. 11. Control performance of TRFN designed by HGAPSO in Example 3, where the reference output is denoted by a solid curve and the actual output by a dotted curve. (a) Output 1. (b) Output 2.

TABLE III
PERFORMANCE COMPARISONS FOR DIFFERENT METHODS OF MIMO TRFN CONTROLLER DESIGN FOR DYNAMIC PLANT CONTROL PROBLEM IN EXAMPLE 3

	Example 3			
Method	GA (Pc = 0.5)	GA (Pc = 0.8)	PSO	HGAPSO
RMSE(Ave)	0.2295	0.3033	0.1770	0.1145
RMSE(Best)	0.1459	0.1764	0.0792	0.0686

see that both the averaged and best RMSEs of HGAPSO are smaller than those of GA and PSO.

VI. CONCLUSION

In this paper, recurrent network design by a new evolutionary algorithm, HGAPSO, is proposed. HGAPSO introduces the concept of the maturing phenomenon in nature into the evolution of individuals originally modeled by GA. The maturing phenomenon is mimicked by PSO, where individuals enhance themselves based on social interactions and their private cognition. From the perspective of PSO, HGAPSO introduces the crossover operation into the society. Thus, the evolution of individuals is no longer restricted to be in the same generation, and better-performed individuals may produce offspring to replace those with poor performance. Thus HGAPSO combines the new individual generation function of both GA and PSO, which together mimics social behaviors of animals, breeding, and survival of the fittest. To demonstrate the performance of HGAPSO, designs of recurrent neural and recurrent fuzzy network are simulated. The results in temporal sequence production by FCRNN and dynamic plant control problems by TRFN demonstrate the superiority of HGAPSO over GA and PSO.

REFERENCES

[1] T. Bäck and H. P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Comput.*, vol. 1, no. 1, pp. 1–23, 1993.

[2] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 1995.

[3] X. Yao, Ed., *Evolutionary Computation: Theory and Applications*, Singapore: World Scientific, 1999.

[4] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[5] J. K. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[6] L. J. Fogel, "Evolutionary programming in perspective: The top-down view," in *Computational Intelligence: Imitating Life*, J. M. Zurada, R. J. Marks II, and C. Goldberg, Eds. Piscataway, NJ: IEEE Press, 1994.

[7] I. Rechenberg, "Evolution strategy," in *Computational Intelligence: Imitating Life*, J. M. Zurada, R. J. Marks II, and C. Goldberg, Eds. Piscataway, NJ: IEEE Press, 1994.

[8] D. J. Montana and L. Davis, "Training feedforward networks using genetic algorithms," in *Proc. Int. Conf. Artifical Intelligence*, Detroit, MI, 1989, pp. 762–767.

[9] Y. Ichikawa and T. Sawa, "Neural network application for direct feedback controllers," *IEEE Trans. Neural Networks*, vol. 3, pp. 224–231, Feb. 1992.

[10] D. Whitley, "Genetic algorithms and neural networks," in *Genetic Algorithms Engineering and Computer Science*, G. Winter, J. Periaux, M. Galan, and P. Cuesta, Eds. New York: Wiley, 1995, pp. 191–201.

[11] O. Cordón, F. Herrera, F. Hoffmann, and L. Magdalena, *Genetic Fuzzy Systems: Evolutionary Tunning and Learning of Fuzzy Knowledge Bases*, Singapore: World Scientific, 2001.

[12] ——, "Ten years of genetic fuzzy ssytems: Current framework and new trends," in *Proc. Joint IFSA World Congress and 20th NAFIPS Int. Conf.*, Vancouver, BC, Canada, 2001, pp. 1241–1246.

[13] R. Eberchart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. Int. Sym. Micro Machine and Human Science*, Nagoya, Japan, Oct. 1995, pp. 39–43.

[14] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Networks*, Perth, Australia, Dec. 1995, pp. 1942–1948.

[15] M. Clerc and J. Kennedy, "The particle swarm—Explosion, stability, and convergence in a multidimentional complex space," *IEEE Trans. Evol.Comput.*, vol. 6, pp. 58–73, Feb. 2002.

[16] C. Zhang, H. Shao, and Y. Li, "Particle swarm optimization for evolving artifical network," in *Proc. IEEE Int. Conf. Syst., Man, Cyber.*, vol. 4, 2000, pp. 2487–2490.

[17] A. P. Engelbrecht and A. Ismail, "Training product unit neural networks," *Stability Control: Theory Appl.*, vol. 2, no. 1–2, pp. 59–74, 1999.

[18] R. Mendes, P. Cortez, M. Rocha, and J. Neves, "Particle swarms for feedforward neural network training," in *Proc. Int. Joint Conf. Neural Networks*, 2002, pp. 1895–1899.

[19] C. T. Lin and C. S. G. Lee, *Neural Fuzzy Systems: A Neural-Fuzzy Synergism to Intelligent Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1996, ch. 13.

[20] V. Gorrini and H. Bersini, "Recurrent fuzzy systems," in *Proc. IEEE Int. Conf. Fuzzy Systems*, vol. 1, Orlando, FL, 1994, pp. 193–198.

[21] J. B. Theocharis and G. Vachtsevanos, "Recursive learning algorithms for training fuzzy recurrent models," *Int. J. Intell. Syst.*, vol. 11, no. 12, pp. 1059–1098, 1996.

[22] G. C. Mouzouri and J. M. Mendel, "Dynamic nonsingleton fuzzy logic systems for nonlinear modeling," *IEEE Trans. Fuzzy Syst.*, vol. 5, pp. 199–208, May 1997.

[23] J. Zhang and A. J. Morris, "Recurrent neuro-fuzzy networks for nonlinear process modeling," *IEEE Trans. Neural Networks*, vol. 10, pp. 313–326, Feb. 1999.

[24] P. A. Mastorocostas and J. B. Theocharis, "A recurrent fuzzy-neural model for dynamic system identification," *IEEE Trans. Syst., Man. Cybern., B*, vol. 32, pp. 176–190, Apr. 2002.

[25] C. F. Juang and C. T. Lin, "A recurrent self-organizing neural fuzzy inference network," *IEEE Trans. Neural Networks*, vol. 10, pp. 828–845, Apr. 1999.

[26] C. H. Lee and C. C. Teng, "Identification and control of dynamic systems using recurrent fuzzy neural networks," *IEEE Trans. Fuzzy Syst.*, vol. 8, pp. 349–366, Aug. 2000.

[27] C. F. Juang, "A TSK-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithm," *IEEE Trans. Fuzzy Syst.*, vol. 10, pp. 155–170, Apr. 2002.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, vol. 1, pp. 318–362.

[29] R. J. Williams and D. Zipser, "A learning algorithm for continually running recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, 1989.

[30] B. A. Pearlmutter, “Learning state space trajectories in recurrent neural networks,” *Neural Comput.*, vol. 1, pp. 263–269, 1989.

[31] P. J. Angeline, G. M. Saunders, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks,” *IEEE Trans. Neural Networks*, vol. 5, pp. 54–65, Jan. 1994.

[32] K. W. C. Ku, M. W. Mak, and W. C. Siu, “Adding learning to cellular genetic algorithms for training recurrent neural networks,” *IEEE Trans. Neural Networks*, vol. 10, pp. 239–252, Mar. 1999.

[33] D. Whitley, T. Starkweather, and C. Bogart, “Genetic algorithm and neural networks: Optimizing connections and connectivity,” *Parallel Computing*, vol. 14, pp. 347–361, 1990.

[34] D. Whitley, V. S. Gordon, and K. Mathias, “Lamarckian evolution, the Baldwin effect and function optimization,” in *Proc. Int. Conf. Parallel Problem Solving from Nature*, vol. 866. New York, 1994, pp. 6–15.

[35] P. Merz and B. Freisleben, “Genetic local search for the TSP: New results,” in *Proc. IEEE Int. Conf. Evolutionary Computation*, Indianapolis, Apr. 1997, pp. 159–164.

[36] A. Jaszkiewicz, “Comparison of local search-based metaheuristics on the multiple-objective knapsack problem,” *Found. Comput. Decision Sci.*, vol. 26, pp. 99–120, Aug. 2001.

[37] N. Krasnogor and J. Smith, “A memetic algorithm with self-adaptive local search: TSP as a case study,” in *Proc. Genetic and Evolutionary Computation Conf.*, Las Vegas, NV, July 2000, pp. 987–994.

[38] J. D. Knowles and D. W. Corne, “M-PAES: A memetic algorithm for multiobjective optimization,” in *Proc. Congress on Evolutionary Computation*, vol. 1. Piscataway, NJ, July 2000, pp. 325–332.

[39] H. Ishibuchi, T. Yoshida, and T. Murata, “Balance between genetic algorithm and local search in memetic algorithms for multiobjective permutation flowshop scheduling,” *IEEE Trans. Evol. Comput.*, vol. 7, pp. 204–223, Apr. 2003.

[40] V. Petridis, S. Kazarlis, and A. Papaikonomou, “A genetic algorithm for training recurrent neural networks,” in *Int. Joint Conf. Neural Networks*, 1993, pp. 2706–2709.

[41] T. Fukuda, T. Kohno, and T. Shibata, “Dynamic memory by recurrent neural network and its learning by genetic algorithm,” in *Proc. IEEE Conf. Decision and Control*, Texas, Dec. 1993, pp. 2815–2820.

[42] J. Kennedy, “The particle swarm: Social adaption of knowledge,” in *Proc. IEEE Int. Conf. Evolutionary Computation*, Apr. 1997, pp. 303–308.

[43] D. Adler, “Genetic algorithms and simulated annealing: A marrage proposal,” in *Proc. IEEE Int. Conf. Neural Networks*, vol. 2, San Franciso, CA, 1993, pp. 1104–1109.

[44] L. Tsinas and B. Dachwald, “A combined neural genetic algorithm,” in *Proc. IEEE Int. Conf. Neural Networks*, vol. 1, 1994, pp. 770–774.

[45] V. Petridis, S. Kazarlis, A. Papaikonomou, and A. Filelis, “A hybrid genetic algorithm for training neural networks,” in *Artifical Neural Networks 2*, I. Aleksander and J. Taylor, Eds, Amsterdam, The Netherlands: North Holland, 1992, pp. 953–956.

[46] K. F. MAn, K. S. Tang, and S. Kwong, *Genetic Algorithms*. New York: Springer-Verlag, 1999.

[47] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1999.

[48] J. Arabas, Z. Michalewicz, and J. Mulawka, “GAVaPS—A genetic algorithm with varying population size,” in *Proc. IEEE Int. Conf. on Evolutionary Computation*, Orlando, 1994, pp. 73–78.

[49] R. Tanese, “Distributed genetic algorithm,” in *Proc. Int. Conf. Genetic Algorithms*, 1989, pp. 434–439.

[50] R. J. Collins and D. R. Jefferson, “Selection in massively parallel genetic algorithms,” in *Proc. Int. Conf. Genetic Algorithms*, 1991, pp. 249–256.

[51] D. E. Moriarty and R. Miikkulainen, “Efficient reinforcement learning through symbiotic evolution,” *Mach. Learn.*, vol. 22, pp. 11–32, 1996.

[52] D. Whitely, S. Dominic, R. Das, and C. W. Anderson, “Genetic reinforcement learning for neurocontrol problems,” *Mach. Learn.*, vol. 13, pp. 259–284, 1993.

[53] D. Thierens, “Adaptive mutation rate control schemes in genetic algorithms,” in *Proc. IEEE Int. Conf. Evolutionary Computation*, HI, 2002, pp. 980–985.

[54] S. Tsutsui and D. E. Goldberg, “Simplex crossover and linkage identification: Single-stage evolution vs. multi-stage evolution,” in *Proc. IEEE Int. Conf. Evolutionary Computation*, HI, 2002, pp. 974–979.

[55] Y. Shi and R. Eberhart, “A modified particle swarm optimizer,” in *Proc. IEEE World Congress on Computational Intelligence*, May 1998, pp. 69–73.

[56] C. L. Karr, “Design of an adaptive fuzzy logic controller using a genetic algorithm,” in *Proc. Int. Conf. Genetic Algorithms*, 1991, pp. 450–457.

[57] A. Homaifar and E. McCormick, “Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms,” *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 2, pp. 129–139, 1995.

[58] K. S. Narendra and K. Parthasarathy, “Identification and control of dynamical systems using neural networks,” *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, Mar. 1990.

Chia-Feng Juang (M’00) received the B.S. and Ph.D. degrees in control engineering from the National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1993 and 1997, respectively.

From 1999 to 2001, he was an Assistant Professor of the Department of Electrical Engineering at the Chung Chou Institute of Technology. In 2001, he joined the National Chung Hsing University, Taichung, Taiwan, R.O.C., where he is currently an Associate Professor of Electrical Engineering. His current research interests are soft computing, intelligent control, computer vision, speech signal processing, and FPGA.

Dr. Juang is a Member of the IEEE Neural Networks Society, the IEEE Signal Processing Society, and the IEEE Systems, Man, and Cybernetics Society.